Introduction to

SRGP

Simple Raster Graphics Package

CSC 292:
Computer Graphics

Randal C. Nelson

Overview of SRGP

- Simple example of a *graphics system*
- Set of basic (C language) routines providing simple 2-D raster display operations, and a simple, user interface for handling interaction.
- Defines structures for 2D geometric primitives such as points and rectangles.
- Provides display operations for drawing (scan converting) lines, circles, rectangles, polygons, etc.; with various widths, colors and fill patterns; text input; clipping etc.
- Interaction handling provides logical abstractions, and sampled and event-driven interaction modes for keyboard and locator devices.

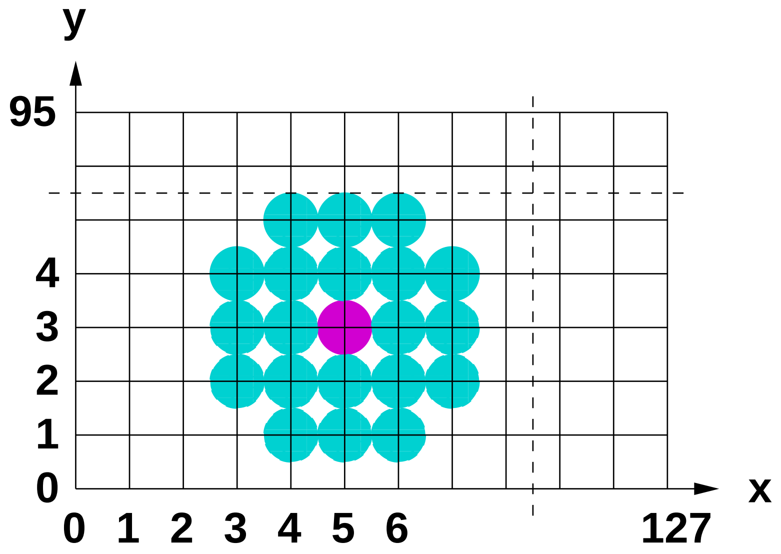
Setup

SRGP draws and interacts on a raster array called a *canvas* which shows up as a window on the user terminal. There can also be other, hidden canvases.

Pixels are indexed by a 2D, (x,y) array, with the (0,0) pixel in the lower lefthand corner.

Only integer index values are allowed

Conceptually, pixels are *centered* on the grid points.



Defined structures

SRGP provides structure definitions for a few, commonly used objects, e.g. points and rectangles.

```
typedef struct{
  int x, y;
} point;

typedef struct{
  point bottomLeft, topRight;
} rectangle;
```

These structures are passed as arguments to and returned by certain routines *directly* (not as pointers), e.g.

```
point1 = SRGP_defPoint(x1, y1);
```

Lineal primitives

SRGP provides routines for drawing various lineal objects, including:

- Line segments, e.g.

```
SRGP_lineCoord(x1, y1, x2, y2);
SRGP_line(point1, point2);
```

- Polylines, e.g.

```
SRGP_polyLineCoord(n_points,
    x_array, y_array);
SRGP_polyLine(
    n_points, point_array);
```

- Polygons and rectangles, e.g.

```
SRGP_polygon(
    n_points, point_array);
SRGP_rectangleCoord(
    left_x, bottom_y,
    right_x, top_y);
```

- Circles and ellipses and arcs, e.g.

```
SRGP_ellipse(
    (bounding_rectangle);
SRGP_ellipseArc(
    bounding_rectangle,
    start_angle, end_angle);
```

Lineal Attributes

By default, lineal objects are one pixel wide and solid, but these *attributes* can be changed by calls to appropriate routines, e.g.

```
SRGP_setLineWidth(5);
SRGP_setLineStyle(DASHED);
SRGP_setColor(color_index);
```

- Line widths are in pixels.
- Line style are members of an *enumerated type*, and there are others (CONTINUOUS, SOLID, ...)
- The color setting routine applies to all operations, and argument is an integer offset into a *color table*. Color definitions *must* be set up by user (see manual).
- Attribute changes remain in effect until changed.

Filled Primitives and Attributes

SRGP also allows primitives such as rectangles, polygons, and ellipses to be filled with colors or patterns, e.g.

```
SRGP_fillPolygon(
    n_points, point_array);
SRGP_fillEllipse(
    bounding_rectangle);
```

There are various routines for setting the fill attributes, e.g.

```
SRGP_setFillStyle(
    BITMAP_PATTERN_OPAQUE);
SRGP_setFillBitmapPattern(
    pattern_index);
```

You can also set the *pen pattern* for lines. See manual for description of default patterns.

Text

In general, text is complicated, due to the number of factors that can vary (font, size, line spacing, character spacing, angle, etc.) SRGP provides fairly simple mechanisms.

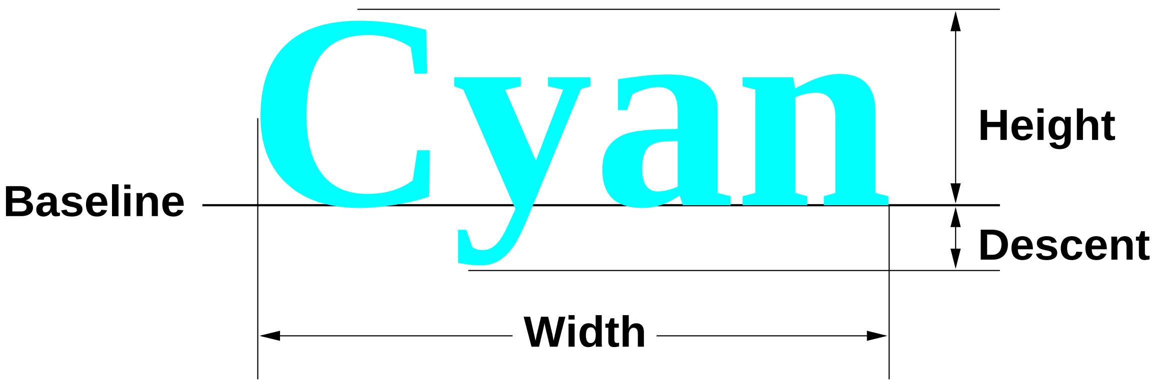
To generate text

```
SRGP_text(
    origin_point, text_string);
```

To change the font (see manual for how to load different fonts)

```
SRGP_setFont(font_index);
```

Sometimes you need to know the dimensions of a string before you write it, e.g. for centering in a button. Several dimensional parameters are associated with a test string as shown below



To find out the dimensions of a string

```
SRGP_inquireTextExtent(
    text_str, *width,
    *height, *descent);
```

Note that *pointers* to the parameters are passed, to allow writeback of values.

Interaction Handling

Interaction involves dealing with a user, and hence following some general human factors guidelines:

- Provide *simple* and *consistent* interaction sequences.
- *Do not overload the user* with too many options.
- *Show available options clearly* at every stage.
- *Give appropriate feedback* to the user.
- Allow users to *recover gracefully* from mistakes.

Logical Input Devices

Idea is to provide some degree of hardware independence by abstracting the *function* of the device from its physical implementation.

SRGP provides support for *locator* (mouse, touch screen) and *keyboard* devices.

Other possibilities

- *Stroke* device (returns polyline of cursor positions).
- *Choice* device (abstracted function keypad).
- *Valuator* device (abstracted knob or slider).

Interaction Modes

Useful distinction can be made between *sampling* or *polling* and *interrupt-driven* processing.

Sampling:

- Loop until some value changes
- Simple, but costly in CPU cycle

Interrupt-driven interaction:

- Use hardware interrupt capability.
- When *event trigger* (e.g. a mouse click) occurs, call interrupt, handler asynchronously.
- Efficient, but complex (very complex).

To avoid complexity of true interrupt-driven control, SRGP (and other graphics packages) offer a poor man's version known as *event-driven* interaction.

- Trigger events (and some associated information) are queued as they occur.
- User program examines the queue and deals with the events at its leisure.
- If there is nothing in the queue, the program can wait *efficiently* until something happens or it times out.
- Offers the simplicity of sampling and the efficiency of interrupt-driven interaction.

Basic interaction concepts in SRGP

- The device *mode* indicates what interaction mode a device is being used in. Can be INACTIVE, SAMPLE, or EVENT and is set by a routine call, e.g.

```
SRGP_setInputMode(LOCATOR, SAMPLE);
```

- In event mode, the event queue is examined by performing a *wait* operation, which removes the first event from the queue (when it appears) and returns the identity of the interrupting device. Returns immediately if the queue is non-empty, and NO_DEVICE if it times out.

```
device = SRGP_waitEvent(max_time);
```

- The device *measure* is the information returned by a device, e.g. characters or strings for a keyboard, and location and buttons pressed for a mouse. Defined by different structures for different devices.

Locator Device

The locator device can be operated in both sample and event mode. The locator measure is defined by the structure

```
typedef struct{
  point position;
  int buttonOfMostRecentTransition;
  enum{UP, DOWN}
      buttonChord[MAX_BUTTON_COUNT];
} locatorMeasure
```

In sample mode, the measure is obtained by the call

```
SRGP_sampleLocator(*measure);
```

In event mode, if the locator is the event trigger, the measure is obtained by the call

```
SRGP_getLocator(*measure);
```

Note that a *pointer* to the measure is passed in.

Keyboard Device

The keyboard device can technically be operated in sample mode, but almost all use is in event mode, to obtain characters or strings.

The keyboard is placed either in RAW mode (characters) or EDIT mode (strings) with the call

```
SRGP_setKeyboardProcessingMode(
    keyboardMode);
```

The measure (char or string) is obtained by the call

```
SRGP_get_Keyboard(
    string, max_string_length);
```

Canvases

Graphic images in SRGP are stored in *Canvases*, which are essentially pixel arrays plus various associated control information.

The *screen canvas* is created at startup, and is the only canvas that is ever visible. (Others must be copied to it to be seen).

Other canvases can be created with the call

```
canvas_id = SRGP_createCanvas(
    width, height);
```

One canvas at a time is *active* (writeable). A canvas is made the active canvas with the call

```
SRGP_useCanvas(canvas_id);
```

Auxiliary canvases are useful for creating internal windows, pulldown menus, etc.

Copying Regions

SRGP provides a powerful pixel copying routine for moving blocks of pixels back and forth between canvases efficiently. The basic call format is

```
SRGP_copyPixel(source_canvas,
     source_rect, dest_point);
```

The copy command can also AND, OR, XOR (and REPLACE) the source block with the destination. This can be useful for various highlighting or combining operations. The mode is set with the call

```
SRGP_setWriteMode(write_mode);
```

Such copy routines are used extensively in implementing window interaction systems, and are generally implemented with the most efficient possible hardware instructions.

Limitation of SRGP

SRPG has plenty of limitations, among them

- No 3D
- Provides only integer coordinate system
- Just a pixel array – no higher-level objects

Such attributes are added in more sophisticated graphics languages such as PostScript, PHIGS, and OpenGL.